autogluon_each_location

October 18, 2023

[1]:
```
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*30
presets = 'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

use_groups = False
n_groups = 8

auto_stack = False
num_stack_levels = 0
num_bag_folds = 8
num_bag_sets = 20

use_tune_data = True
use_test_data = True
tune_and_test_length = 0.5 # 3 months from end
holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
 ↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False
sample_weight_estimated = 1

run_analysis = True
```

[2]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")


def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,␣
 ↪we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")


    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
       'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
       'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    X_shifted = X[X.index.minute==0][columns].copy()
    # loop through all rows and check if index + 1 hour is in the index, if so␣
 ↪get that value, else nan
    count1 = 0
    count2 = 0
    for i in range(len(X_shifted)):
        if X_shifted.index[i] + pd.Timedelta('1 hour') in X.index:
            count1 += 1
            X_shifted.iloc[i] = X.loc[X_shifted.index[i] + pd.Timedelta('1␣
 ↪hour')][columns]
        else:
            count2 += 1
            X_shifted.iloc[i] = np.nan

    print("COUNT1", count1)
    print("COUNT2", count2)

    X_old_unshifted = X[X.index.minute==0][columns]
    # rename X_old_unshifted columns to have _not_shifted at the end
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
 ↪columns]

    # put the shifted columns back into the original dataframe
    #X[columns] = X_shifted[columns]



    date_calc = None
    if "date_calc" in X.columns:
        date_calc = X[X.index.minute == 0]['date_calc']

    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X




def fix_X(X, name):
    # Convert 'date_forecast' to datetime format and replace original column␣
 ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X



def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
 ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
 ↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -␣
 ↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
        X_test["estimated_diff_hours"] = (X_test.index - pd.
 ↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] =␣
 ↪X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
 ↪fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1

    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)
    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,␣
↪right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
```

```
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
```

1 Feature enginering

[3]:
```
import numpy as np
import pandas as pd

X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
 ↪inplace=True)
```

```
for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

print(X_train.head())
```

```
if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name␣
 ↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),␣
 ↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())
```

```
to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",␣
 ↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",␣
 ↪"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",␣
 ↪"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
 ↪gm3", "air_density_2m:kgm3"]
```

```python
X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 22:00:00                     7.700              1.22825
2019-06-02 23:00:00                     7.700              1.22350
2019-06-03 00:00:00                     7.875              1.21975
2019-06-03 01:00:00                     8.425              1.21800
2019-06-03 02:00:00                     8.950              1.21800

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
2019-06-02 22:00:00           1728.949951               0.000000
2019-06-02 23:00:00           1689.824951               0.000000
2019-06-03 00:00:00           1563.224976               0.000000
2019-06-03 01:00:00           1283.425049            6546.899902
2019-06-03 02:00:00           1003.500000          102225.898438

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 22:00:00             0.00       1728.949951              0.0
2019-06-02 23:00:00             0.00       1689.824951              0.0
2019-06-03 00:00:00             0.00       1563.224976              0.0
2019-06-03 01:00:00             0.75       1283.425049              0.0
2019-06-03 02:00:00            23.10       1003.500000              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                   …
2019-06-02 22:00:00      280.299988          0.000          0.000000  …
2019-06-02 23:00:00      280.299988          0.000          0.000000  …
2019-06-03 00:00:00      280.649994          0.000          0.000000  …
2019-06-03 01:00:00      281.674988          0.300       7743.299805  …
2019-06-03 02:00:00      282.500000         11.975      60137.601562  …

                     t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 22:00:00   286.225006           100.000000  40386.476562
2019-06-02 23:00:00   286.899994           100.000000  33770.648438
2019-06-03 00:00:00   286.950012           100.000000  13595.500000
2019-06-03 01:00:00   286.750000           100.000000   2321.850098
2019-06-03 02:00:00   286.450012            99.224998  11634.799805

                     wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 22:00:00                3.600                -3.575
2019-06-02 23:00:00                3.350                -3.350
2019-06-03 00:00:00                3.050                -2.950
2019-06-03 01:00:00                2.725                -2.600
2019-06-03 02:00:00                2.550                -2.350

                     wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 22:00:00               -0.500                      0.0
2019-06-02 23:00:00                0.275                      0.0
2019-06-03 00:00:00                0.750                      0.0
2019-06-03 01:00:00                0.875                      0.0
2019-06-03 02:00:00                0.925                      0.0

                     is_estimated      y  location
ds
2019-06-02 22:00:00             0   0.00         A
2019-06-02 23:00:00             0   0.00         A
2019-06-03 00:00:00             0   0.00         A
2019-06-03 01:00:00             0   0.00         A
2019-06-03 02:00:00             0  19.36         A

[5 rows x 48 columns]
```

[4]:
```
def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /␣
↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

import pandas as pd
import numpy as np

def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the␣
↪bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
        frac1 (float): The fraction of each bin to go into the first output␣
↪dataset.
```

```
    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Calculate bin size
    bin_size = len(input_data) // num_bins

    # Initialize empty DataFrames for output
    output_data1 = pd.DataFrame()
    output_data2 = pd.DataFrame()

    for i in range(num_bins):
        # Shuffle the data in the current bin
        np.random.seed(i)
        current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].
↪sample(frac=1)

        # Calculate the sizes for each output set
        size1 = int(len(current_bin) * frac1)

        # Split and append to output DataFrames
        output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
        output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

    # Shuffle and split the remaining data
    remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)
    remaining_size1 = int(len(remaining_data) * frac1)

    output_data1 = pd.concat([output_data1, remaining_data.iloc[:
↪remaining_size1]])
    output_data2 = pd.concat([output_data2, remaining_data.iloc[remaining_size1:
↪]])

    return output_data1, output_data2
```

[5]:
```
from autogluon.tabular import TabularDataset, TabularPredictor
from autogluon.timeseries import TimeSeriesDataFrame
import numpy as np
```

```
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the␣
 ↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
 ↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
test_set = TabularDataset(data[data["ds"] >= split_time])

# shuffle test_set and only grab tune_and_test_length percent of it, rest goes␣
 ↪to train_set
test_set, new_train_set = split_and_shuffle_data(test_set, 40,␣
 ↪tune_and_test_length)


print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))
print("Length of test set", len(test_set))




if use_groups:
    test_set = test_set.drop(columns=['group'])


tuning_data = None
if use_tune_data:
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            # randomly shuffle the loc_test_set
            loc_tuning_data, loc_test_data =␣
↪split_and_shuffle_data(loc_test_set, 40, 0.5)
            tuning_data.append(loc_tuning_data)
            test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.
↪shape[0], tuning_data.shape[0] + test_data.shape[0])

    else:
        tuning_data = test_set
        print("Shape of tuning", tuning_data.shape[0])

    # ensure sample weights for your tuning data sum to the number of rows in␣
↪the tuning data.
    if weight_evaluation:
        tuning_data = normalize_sample_weights_per_location(tuning_data)


else:
    if use_test_data:
        test_data = test_set
        print("Shape of test", test_data.shape[0])


train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of␣
↪rows in the training (or tuning) data.
if weight_evaluation:
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
if use_tune_data:
    tuning_data = TabularDataset(tuning_data)
if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Length of train set before adding test set 82026
Length of train set after adding test set 87486
Length of test set 5459
Shapes of tuning and test 2728 2731 5459
```

```
[6]: if run_analysis:
         import autogluon.eda.auto as auto
         auto.dataset_overview(train_data=train_data, test_data=test_data,␣
      ↪label="y", sample=None)
```

```
[7]: if run_analysis:
         auto.target_analysis(train_data=train_data, label="y", sample=None)
```

2 Starting

```
[8]: import os


     # Get the last submission number
     last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
      ↪filename in os.listdir('submissions') if "submission" in filename]))
     print("Last submission number:", last_submission_number)
     print("Now creating submission number:", last_submission_number + 1)

     # Create the new filename
     new_filename = f'submission_{last_submission_number + 1}'

     hello = os.environ.get('HELLO')
     if hello is not None:
         new_filename += f'_{hello}'

     print("New filename:", new_filename)
```

```
Last submission number: 94
Now creating submission number: 95
New filename: submission_95
```

```
[9]: predictors = [None, None, None]
```

```
[10]: def fit_predictor_for_location(loc):
          print(f"Training model for location {loc}...")
          # sum of sample weights for this location, and number of rows, for both␣
       ↪train and tune data and test data
          if weight_evaluation:
              print("Train data sample weight sum:",␣
       ↪train_data[train_data["location"] == loc]["sample_weight"].sum())
              print("Train data number of rows:", train_data[train_data["location"]␣
       ↪== loc].shape[0])
              if use_tune_data:
                  print("Tune data sample weight sum:",␣
       ↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
```

```
            print("Tune data number of rows:",␣
↪tuning_data[tuning_data["location"] == loc].shape[0])
        if use_test_data:
            print("Test data sample weight sum:",␣
↪test_data[test_data["location"] == loc]["sample_weight"].sum())
            print("Test data number of rows:", test_data[test_data["location"]␣
↪== loc].shape[0])
    predictor = TabularPredictor(
        label=label,
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc].
↪drop(columns=["ds"]),
        time_limit=time_limit,
        # presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
↪somethin, will be overwritten anyways
        num_bag_sets=num_bag_sets,
        tuning_data=tuning_data[tuning_data["location"] == loc].
↪reset_index(drop=True).drop(columns=["ds"]) if use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Warning: path already exists! This predictor may overwrite an existing
predictor! path="AutogluonModels/submission_95_A"
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_95_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   225.79 GB / 315.93 GB (71.5%)
Train Data Rows:    31872
Train Data Columns: 32
Tuning Data Rows:    1093
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 649.68162,
1178.37671)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    132099.78 MB
        Train Data (Original)  Memory Usage: 10.09 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…

Training model for location A…

        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 7.68 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.15s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 1799.85s of
the 1799.85s of remaining time.
        -140.7608        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.37s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 1799.35s of
the 1799.35s of remaining time.
```

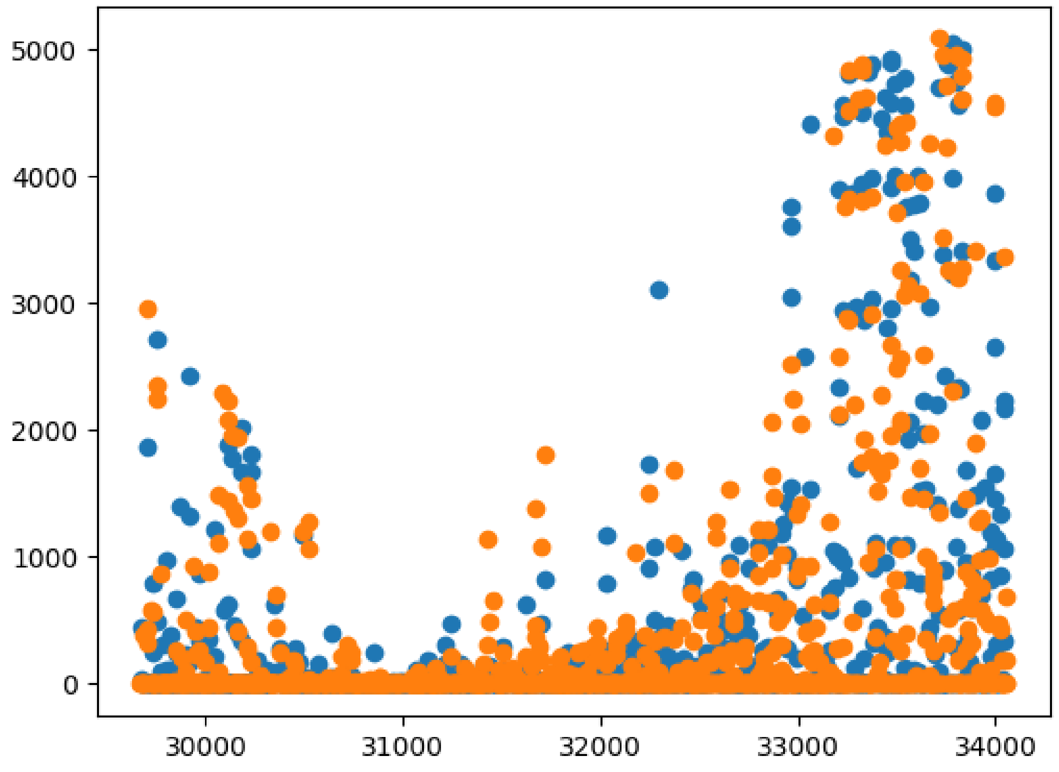
```
        -140.9566        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        1.77s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1797.49s of the
1797.49s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -94.2003         = Validation score   (-mean_absolute_error)
        28.57s   = Training   runtime
        17.78s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1759.05s of the
1759.04s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -96.9911         = Validation score   (-mean_absolute_error)
        22.75s   = Training   runtime
        6.01s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 1732.38s of
the 1732.38s of remaining time.
        -108.2593        = Validation score   (-mean_absolute_error)
        7.8s     = Training   runtime
        1.15s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1722.18s of the
1722.18s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -104.4091        = Validation score   (-mean_absolute_error)
        193.94s  = Training   runtime
        0.1s     = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 1527.07s of the
1527.07s of remaining time.
        -111.4972        = Validation score   (-mean_absolute_error)
        1.57s    = Training   runtime
        1.16s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 1523.09s of
the 1523.08s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -108.7347        = Validation score   (-mean_absolute_error)
        38.32s   = Training   runtime
        0.52s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 1483.04s of the
1483.04s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -103.0986        = Validation score   (-mean_absolute_error)
        5.81s    = Training   runtime
        0.29s    = Validation runtime
```

```
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 1475.22s of
the 1475.22s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -95.6392         = Validation score   (-mean_absolute_error)
        122.2s   = Training   runtime
        0.32s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 1351.62s of the
1351.61s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -96.389  = Validation score   (-mean_absolute_error)
        91.51s   = Training   runtime
        20.11s   = Validation runtime
Repeating k-fold bagging: 2/20
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1250.05s of the
1250.05s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -93.9626         = Validation score   (-mean_absolute_error)
        57.39s   = Training   runtime
        37.52s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1214.21s of the
1214.21s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -97.3732         = Validation score   (-mean_absolute_error)
        47.68s   = Training   runtime
        11.53s   = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1184.68s of the
1184.68s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -103.9853        = Validation score   (-mean_absolute_error)
        386.86s  = Training   runtime
        0.2s     = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 990.4s of the
990.4s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -109.1851        = Validation score   (-mean_absolute_error)
        77.28s   = Training   runtime
        0.99s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 949.01s of the
949.01s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -102.09  = Validation score   (-mean_absolute_error)
        13.4s    = Training   runtime
        0.67s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 939.7s of the
939.7s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -94.3911         = Validation score   (-mean_absolute_error)
        240.91s  = Training   runtime
        0.62s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 819.44s of the
819.44s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -95.8798         = Validation score   (-mean_absolute_error)
        182.46s  = Training   runtime
        41.61s   = Validation runtime
Completed 2/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
714.62s of remaining time.
        -90.3057         = Validation score   (-mean_absolute_error)
        0.44s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 1085.85s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_95_A/")
Evaluation: mean_absolute_error on test data: -89.73617380614655
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -89.73617380614655,
    "root_mean_squared_error": -302.74105040128256,
    "mean_squared_error": -91652.14359807191,
    "r2": 0.9022650490620234,
    "pearsonr": 0.9507610033290933,
    "median_absolute_error": -2.8489151000976562
}

Evaluation on test data:
-89.73617380614655
```

[11]:
```
import matplotlib.pyplot as plt

leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_test_data:
        lb = predictors[i].leaderboard(test_data[test_data["location"] == loc])
        lb["location"] = loc
        plt.scatter(test_data[test_data["location"] == loc]["y"].index,␣
 ↪test_data[test_data["location"] == loc]["y"])
        if use_tune_data:
            plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,␣
 ↪tuning_data[tuning_data["location"] == loc]["y"])
        plt.show()

        return lb
    else:
        return pd.DataFrame()

leaderboards[0] = leaderboard_for_location(0, loc)
```

```
                     model  score_test   score_val  pred_time_test
pred_time_val    fit_time  pred_time_test_marginal  pred_time_val_marginal
fit_time_marginal  stack_level  can_infer  fit_order
0      WeightedEnsemble_L2  -89.736174  -90.305682        3.143599
38.143281  298.740003                 0.003387                0.000630
0.438172            2       True         12
1        LightGBMXT_BAG_L1  -93.483327  -93.962592        2.765459
37.524019   57.391981                 2.765459               37.524019
57.391981            1       True          3
2     NeuralNetTorch_BAG_L1  -95.030981  -94.391102        0.374753
0.618632  240.909850                 0.374753                0.618632
240.909850            1       True         10
3      LightGBMLarge_BAG_L1  -96.097045  -95.879821        7.701581
41.606520  182.459504                 7.701581               41.606520
182.459504            1       True         11
4          LightGBM_BAG_L1  -97.662003  -97.373181        1.597548
11.531363   47.680326                 1.597548               11.531363
47.680326            1       True          4
5          CatBoost_BAG_L1 -102.251230 -103.985266        0.141490
0.196846  386.859108                 0.141490                0.196846
386.859108            1       True          6
6   RandomForestMSE_BAG_L1 -102.391903 -108.259330        0.627777
1.150095    7.798758                 0.627777                1.150095
7.798758            1       True          5
7     ExtraTreesMSE_BAG_L1 -103.794354 -111.497167        0.636150
1.157234    1.574136                 0.636150                1.157234
1.574136            1       True          7
8           XGBoost_BAG_L1 -104.486566 -102.089956        0.278424
0.667368   13.397035                 0.278424                0.667368
13.397035            1       True          9
9   NeuralNetFastAI_BAG_L1 -107.250101 -109.185143        1.057474
0.991382   77.281300                 1.057474                0.991382
77.281300            1       True          8
```

```
10   KNeighborsDist_BAG_L1 -124.177226 -140.956605        0.020517
1.768634    0.029982                  0.020517              1.768634
0.029982            1        True           2
11   KNeighborsUnif_BAG_L1 -124.918514 -140.760811        0.233926
0.371370    0.030955                  0.233926              0.371370
0.030955            1        True           1
```



[12]:
```python
loc = "B"
predictors[1] = fit_predictor_for_location(loc)
leaderboards[1] = leaderboard_for_location(1, loc)
```

```
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_95_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   222.59 GB / 315.93 GB (70.5%)
Train Data Rows:    31020
Train Data Columns: 32
Tuning Data Rows:    898
Tuning Data Columns: 32
```

```
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 99.56591, 196.469)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …

Training model for location B…

Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    130195.25 MB
        Train Data (Original)  Memory Usage: 9.77 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 7.44 MB (0.0% of available memory)
```

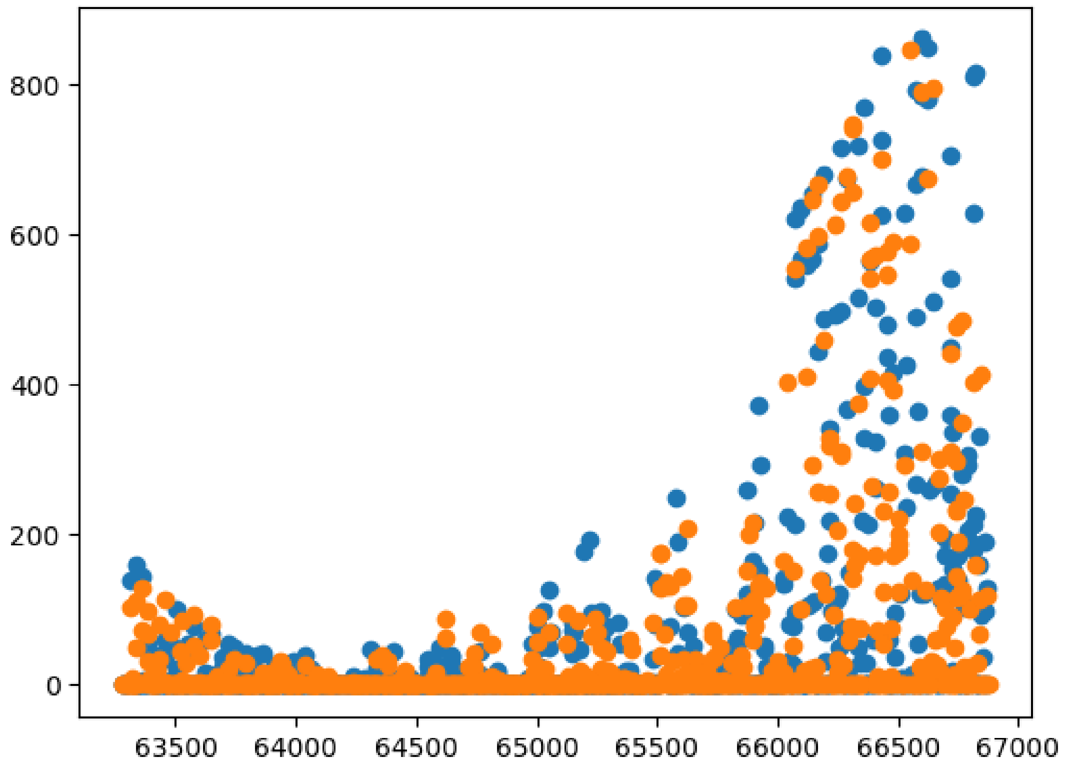
```
Data preprocessing and feature engineering runtime = 0.16s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 1799.84s of
the 1799.84s of remaining time.
        -23.6782         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.38s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 1799.37s of
the 1799.36s of remaining time.
        -23.6491         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.37s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1798.9s of the
1798.9s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -15.2374         = Validation score   (-mean_absolute_error)
        29.86s   = Training   runtime
        19.91s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1763.84s of the
```

```
1763.84s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -15.1634         = Validation score   (-mean_absolute_error)
        30.66s   = Training   runtime
        15.76s   = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 1728.1s of
the 1728.09s of remaining time.
        -15.7551         = Validation score   (-mean_absolute_error)
        8.64s    = Training   runtime
        1.11s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1717.25s of the
1717.25s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -16.2585         = Validation score   (-mean_absolute_error)
        192.85s  = Training   runtime
        0.1s     = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 1523.08s of the
1523.07s of remaining time.
        -14.8929         = Validation score   (-mean_absolute_error)
        1.5s     = Training   runtime
        1.14s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 1519.28s of
the 1519.28s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.2324         = Validation score   (-mean_absolute_error)
        37.85s   = Training   runtime
        0.47s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 1479.65s of the
1479.65s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -15.2878         = Validation score   (-mean_absolute_error)
        82.93s   = Training   runtime
        23.92s   = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 1390.56s of
the 1390.56s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -11.4203         = Validation score   (-mean_absolute_error)
        177.81s  = Training   runtime
        0.32s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 1211.36s of the
1211.36s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
```

```
        -14.1699         = Validation score   (-mean_absolute_error)
        96.14s   = Training   runtime
        22.06s   = Validation runtime
Repeating k-fold bagging: 2/20
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1104.1s of the
1104.1s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -15.3672         = Validation score   (-mean_absolute_error)
        59.6s    = Training   runtime
        37.15s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1068.1s of the
1068.09s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -15.1239         = Validation score   (-mean_absolute_error)
        61.35s   = Training   runtime
        31.65s   = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1030.85s of the
1030.85s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -16.2743         = Validation score   (-mean_absolute_error)
        384.52s  = Training   runtime
        0.2s     = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 837.91s of
the 837.91s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.0157         = Validation score   (-mean_absolute_error)
        75.69s   = Training   runtime
        0.94s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 797.68s of the
797.68s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -15.1725         = Validation score   (-mean_absolute_error)
        164.62s  = Training   runtime
        52.25s   = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 707.25s of the
707.25s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -11.1896         = Validation score   (-mean_absolute_error)
        343.18s  = Training   runtime
        0.64s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 540.36s of the
540.36s of remaining time.
```

```
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.2045         = Validation score   (-mean_absolute_error)
        191.71s  = Training   runtime
        42.69s   = Validation runtime
Completed 2/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
431.0s of remaining time.
        -11.1589         = Validation score   (-mean_absolute_error)
        0.42s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 1369.44s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_95_B/")
Evaluation: mean_absolute_error on test data: -14.406410052388141
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -14.406410052388141,
    "root_mean_squared_error": -43.452274019795816,
    "mean_squared_error": -1888.1001174914227,
    "r2": 0.9146483579094978,
    "pearsonr": 0.9570577027925582,
    "median_absolute_error": -0.24670492112636566
}

Evaluation on test data:
-14.406410052388141
                     model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0    NeuralNetTorch_BAG_L1  -14.394115 -11.189584        0.358684       0.643370
343.177381                0.358684                0.643370         343.177381
1       True         10
1      WeightedEnsemble_L2  -14.406410 -11.158949        5.245321      54.038878
509.717989                0.003725                0.000669           0.415344
2       True         12
2   NeuralNetFastAI_BAG_L1  -16.266971 -14.015705        1.020539       0.941621
75.688813                1.020539                0.941621          75.688813
1       True          8
3     LightGBMLarge_BAG_L1  -16.690501 -14.204470        7.639017      42.687759
191.707300                7.639017               42.687759         191.707300
1       True         11
4           XGBoost_BAG_L1  -17.890860 -15.172511        4.361572      52.253230
164.622181                4.361572               52.253230         164.622181
1       True          9
```

```
5          LightGBM_BAG_L1  -17.992811 -15.123885        2.547344      31.645180
61.352312                 2.547344              31.645180          61.352312
1       True           4
6          CatBoost_BAG_L1  -18.481695 -16.274294        0.140102       0.202582
384.515088                 0.140102              0.202582          384.515088
1       True           6
7        LightGBMXT_BAG_L1  -18.788441 -15.367202        2.757311      37.153268
59.604928                 2.757311              37.153268          59.604928
1       True           3
8      ExtraTreesMSE_BAG_L1  -19.308161 -14.892934        0.521340       1.141609
1.503083                 0.521340              1.141609          1.503083
1       True           7
9    RandomForestMSE_BAG_L1  -19.879611 -15.755074        0.493666       1.114611
8.636562                 0.493666              1.114611          8.636562
1       True           5
10    KNeighborsDist_BAG_L1  -30.037969 -23.649097        0.016531       0.369148
0.029655                 0.016531              0.369148          0.029655
1       True           2
11    KNeighborsUnif_BAG_L1  -30.061031 -23.678158        0.017909       0.384116
0.030099                 0.017909              0.384116          0.030099
1       True           1
```



[13]:
```
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
leaderboards[2] = leaderboard_for_location(2, loc)
```

```
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_95_C/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   218.10 GB / 315.93 GB (69.0%)
Train Data Rows:    24594
Train Data Columns: 32
Tuning Data Rows:    737
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
        Label info (max, min, mean, stddev): (999.6, -0.0, 79.8926, 168.407)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    129989.42 MB

Training model for location C…

        Train Data (Original)  Memory Usage: 7.75 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
```

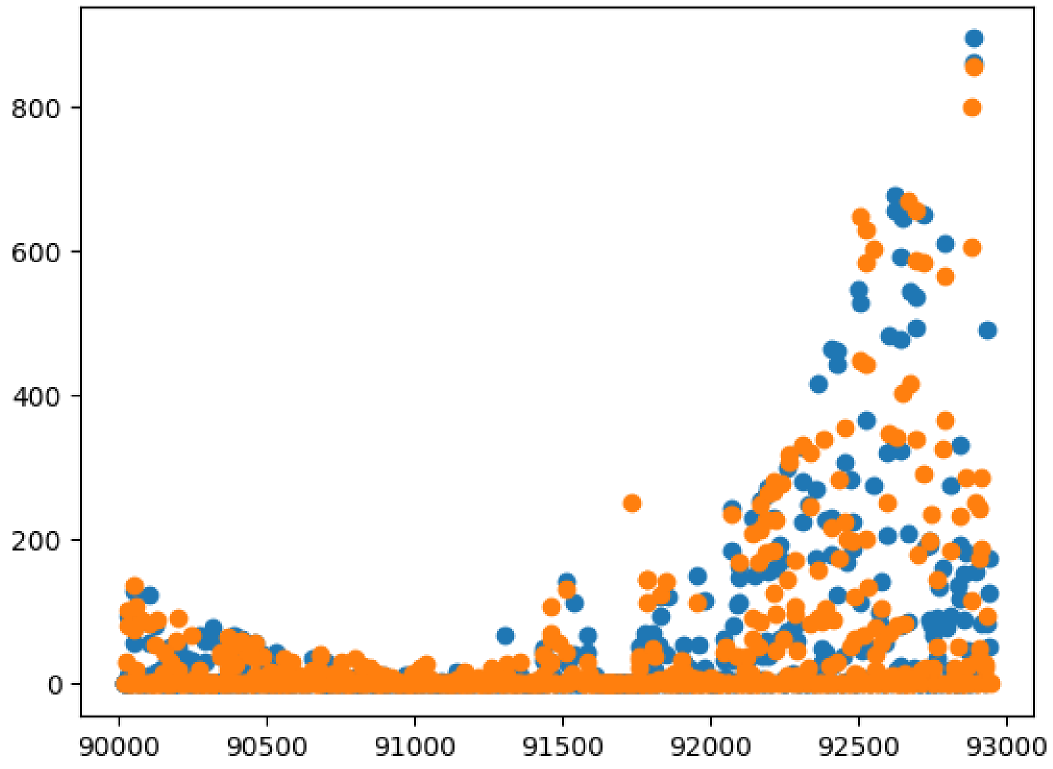
```
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 5.9 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.14s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 1799.86s of
the 1799.85s of remaining time.
```

```
        -23.6472         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.28s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 1799.49s of
the 1799.49s of remaining time.
        -23.6995         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.32s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1798.8s of the
1798.8s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.2571         = Validation score   (-mean_absolute_error)
        25.74s   = Training   runtime
        14.6s    = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1767.83s of the
1767.83s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.7152         = Validation score   (-mean_absolute_error)
        23.63s   = Training   runtime
        5.31s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 1740.38s of
the 1740.38s of remaining time.
        -16.5538         = Validation score   (-mean_absolute_error)
        4.75s    = Training   runtime
        0.75s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1734.27s of the
1734.27s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.2087         = Validation score   (-mean_absolute_error)
        186.66s  = Training   runtime
        0.09s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 1546.33s of the
1546.33s of remaining time.
        -16.5323         = Validation score   (-mean_absolute_error)
        0.97s    = Training   runtime
        0.78s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 1543.89s of
the 1543.89s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.7932         = Validation score   (-mean_absolute_error)
        30.84s   = Training   runtime
        0.41s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 1511.41s of the
1511.41s of remaining time.
```

```
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.642  = Validation score   (-mean_absolute_error)
        59.66s   = Training   runtime
        4.92s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 1448.04s of
the 1448.04s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.1165         = Validation score   (-mean_absolute_error)
        88.69s   = Training   runtime
        0.3s     = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 1357.96s of the
1357.96s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.9832         = Validation score   (-mean_absolute_error)
        90.25s   = Training   runtime
        11.16s   = Validation runtime
Repeating k-fold bagging: 2/20
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1259.84s of the
1259.84s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.3447         = Validation score   (-mean_absolute_error)
        52.06s   = Training   runtime
        28.69s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1227.19s of the
1227.19s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.601  = Validation score   (-mean_absolute_error)
        47.75s   = Training   runtime
        9.36s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1198.78s of the
1198.78s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.1853         = Validation score   (-mean_absolute_error)
        373.37s  = Training   runtime
        0.17s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 1010.81s of
the 1010.81s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.8261         = Validation score   (-mean_absolute_error)
        61.51s   = Training   runtime
        0.82s    = Validation runtime
```

```
Fitting model: XGBoost_BAG_L1 … Training model for up to 978.07s of the
978.07s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.7949         = Validation score   (-mean_absolute_error)
        103.07s  = Training   runtime
        7.04s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 930.53s of the
930.53s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.1899         = Validation score   (-mean_absolute_error)
        169.42s  = Training   runtime
        0.66s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 848.31s of the
848.31s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.8232         = Validation score   (-mean_absolute_error)
        176.54s  = Training   runtime
        22.83s   = Validation runtime
Repeating k-fold bagging: 3/20
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 750.53s of the
750.53s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.3538         = Validation score   (-mean_absolute_error)
        78.19s   = Training   runtime
        43.95s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 717.13s of the
717.13s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.6223         = Validation score   (-mean_absolute_error)
        71.47s   = Training   runtime
        15.41s   = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 688.26s of the
688.26s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.2121         = Validation score   (-mean_absolute_error)
        558.61s  = Training   runtime
        0.26s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 501.68s of
the 501.68s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.8243         = Validation score   (-mean_absolute_error)
        91.97s   = Training   runtime
        1.25s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 468.57s of the
468.57s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.8108         = Validation score   (-mean_absolute_error)
        144.63s  = Training   runtime
        8.26s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 422.15s of the
422.15s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.151  = Validation score   (-mean_absolute_error)
        253.51s  = Training   runtime
        0.96s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 336.35s of the
336.34s of remaining time.
        Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.8272         = Validation score   (-mean_absolute_error)
        266.84s  = Training   runtime
        34.71s   = Validation runtime
Completed 3/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
231.27s of remaining time.
        -11.8134         = Validation score   (-mean_absolute_error)
        0.42s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 1569.84s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_95_C/")
Evaluation: mean_absolute_error on test data: -11.83937215036181
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -11.83937215036181,
    "root_mean_squared_error": -32.75979055599026,
    "mean_squared_error": -1073.2038772723483,
    "r2": 0.9146992864441046,
    "pearsonr": 0.9565666689835377,
    "median_absolute_error": -0.6500986218452454
}

Evaluation on test data:
-11.83937215036181
```

```
                     model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0      WeightedEnsemble_L2  -11.839372 -11.813410       16.551547      80.879610
690.917505                 0.004423                0.000627           0.415635
2       True          12
1        LightGBMXT_BAG_L1  -12.232382 -12.353825        4.669291      43.953349
78.190710                 4.669291               43.953349          78.190710
1       True           3
2     LightGBMLarge_BAG_L1  -12.535522 -12.827229       10.117924      34.708794
266.836030                10.117924               34.708794         266.836030
1       True          11
3          LightGBM_BAG_L1  -12.877325 -13.622306        3.166363      15.410607
71.466252                 3.166363               15.410607          71.466252
1       True           4
4    NeuralNetTorch_BAG_L1  -13.425317 -14.151038        0.524846       0.964426
253.507037                 0.524846                0.964426         253.507037
1       True          10
5          CatBoost_BAG_L1  -13.528117 -13.212063        0.191049       0.257054
558.609798                 0.191049                0.257054         558.609798
1       True           6
6           XGBoost_BAG_L1  -13.591789 -13.810809        2.280030       8.260043
144.626493                 2.280030                8.260043         144.626493
1       True           9
7   NeuralNetFastAI_BAG_L1  -14.342433 -14.824308        1.235063       1.252413
91.968093                 1.235063                1.252413          91.968093
1       True           8
8     ExtraTreesMSE_BAG_L1  -16.166448 -16.532338        0.316099       0.778436
0.974661                 0.316099                0.778436           0.974661
1       True           7
9   RandomForestMSE_BAG_L1  -16.511285 -16.553818        0.291723       0.754177
4.749317                 0.291723                0.754177           4.749317
1       True           5
10   KNeighborsDist_BAG_L1  -23.919331 -23.699494        0.013280       0.324103
0.023947                 0.013280                0.324103           0.023947
1       True           2
11   KNeighborsUnif_BAG_L1  -24.102600 -23.647165        0.013989       0.280483
0.024302                 0.013989                0.280483           0.024302
1       True           1
```



[14]:
```python
# save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

3 Submit

[15]:
```python
import pandas as pd
import matplotlib.pyplot as plt

train_data_with_dates = TabularDataset('X_train_raw.csv')
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

test_data = TabularDataset('X_test_raw.csv')
test_data["ds"] = pd.to_datetime(test_data["ds"])
#test_data
```

```
Loaded data from: X_train_raw.csv | Columns = 34 / 34 | Rows = 92945 -> 92945
Loaded data from: X_test_raw.csv | Columns = 33 / 33 | Rows = 4608 -> 4608
```

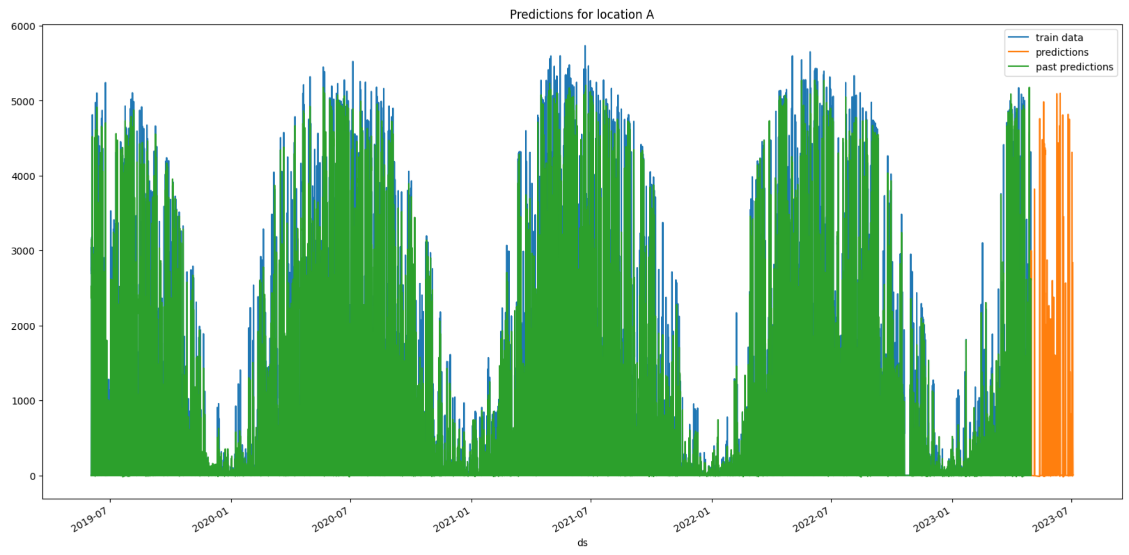
[16]:
```python
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
```

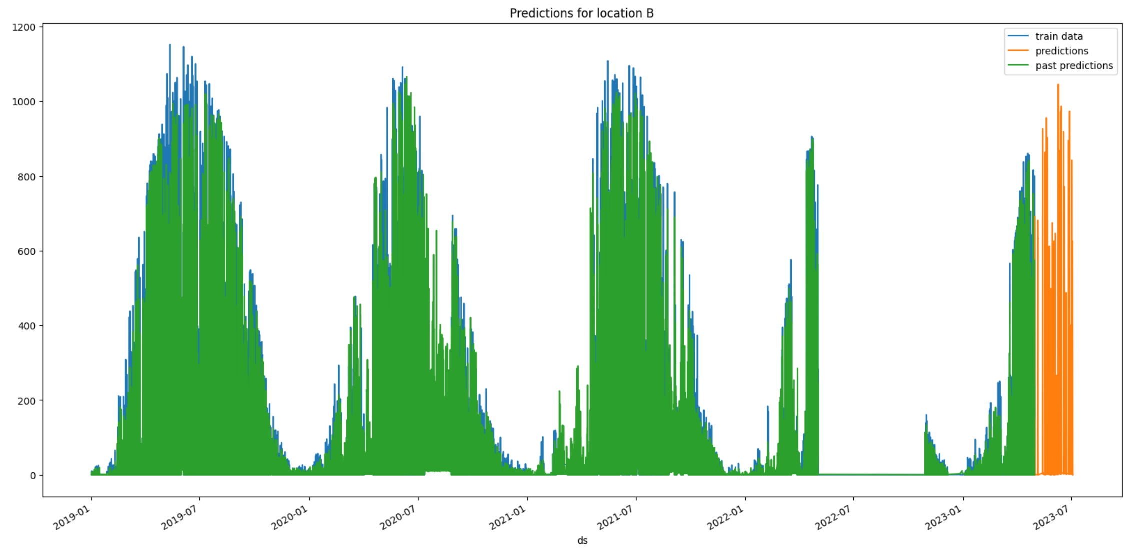
```
test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",␣
↪"location"], left_on=["ds", "location"])

#test_data_merged
```

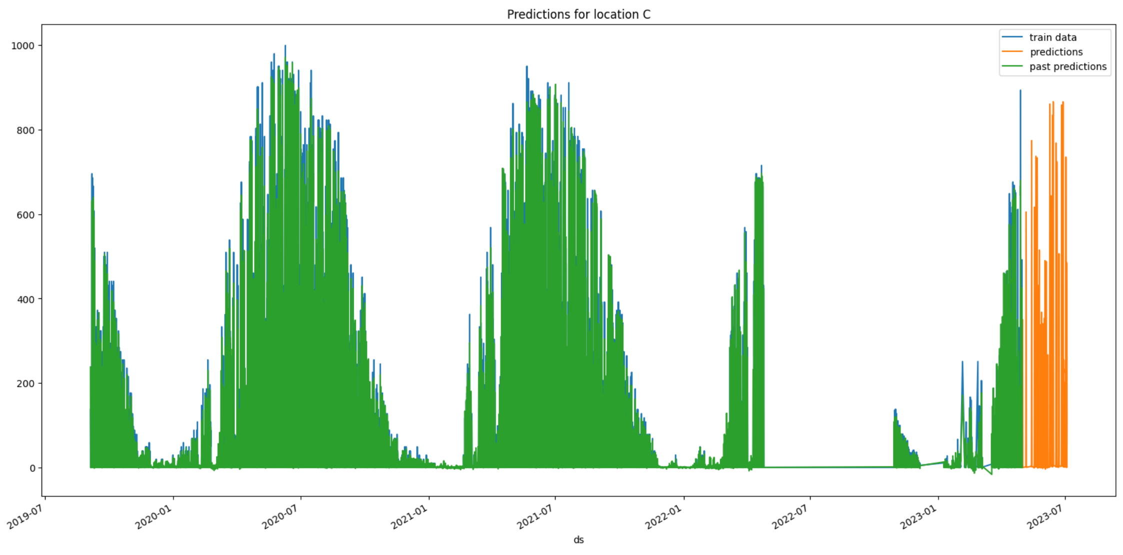
```
Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160
```

[17]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in test_data.groupby('location'):
    i = location_map[loc]
    subset = test_data_merged[test_data_merged["location"] == loc].
↪reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    past_pred = predictors[i].
↪predict(train_data_with_dates[train_data_with_dates["location"] == loc])
    train_data_with_dates.loc[train_data_with_dates["location"] == loc,␣
↪"prediction"] = past_pred
```

[18]:
```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='y', ax=ax, label="train data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='prediction', ax=ax, label="past predictions")

    # title
    ax.set_title(f"Predictions for location {loc}")
```


Predictions for location A
train data
predictions
past predictions
6000
5000
4000
3000
2000
1000
0
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
ds


Predictions for location B
train data
predictions
past predictions
1200
1000
800
600
400
200
0
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
ds



[19]:
```python
# concatenate predictions
submissions_df = pd.concat(predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

[19]:
```
       id   prediction
0       0    -1.181474
1       1    -0.678774
2       2    -1.221545
3       3    58.305767
4       4   308.260925
..    ...          ...
715  2155    69.737434
716  2156    41.630219
717  2157    10.523461
718  2158     4.871428
719  2159     1.108044

[2160 rows x 2 columns]
```

[20]:
```python
# Save the submission DataFrame to submissions folder, create new name based on␣
↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),␣
↪index=False)
print("jall1a")
```

```
Saving submission to submissions/submission_95.csv
jall1a
```

[21]:
```
# save this running notebook
from IPython.display import display, Javascript
import time

# hei123

display(Javascript("IPython.notebook.save_checkpoint();"))

time.sleep(3)
```

```
<IPython.core.display.Javascript object>
```

[22]:
```
# save this notebook to submissions folder
import subprocess
import os
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
 ↪ipynb"])
```

```
[NbConvertApp] Converting notebook autogluon_each_location.ipynb to pdf
/opt/conda/lib/python3.10/site-packages/nbconvert/utils/pandoc.py:51:
RuntimeWarning: You are using an unsupported version of pandoc (2.9.2.1).
Your version must be at least (2.14.2) but less than (4.0.0).
Refer to https://pandoc.org/installing.html.
Continuing with doubts…
  check_pandoc_version()
[NbConvertApp] Support files will be in notebook_pdfs/submission_95_files/
[NbConvertApp] Making directory
./notebook_pdfs/submission_95_files/notebook_pdfs
[NbConvertApp] Writing 145979 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 215812 bytes to notebook_pdfs/submission_95.pdf
```

[22]:
```
CompletedProcess(args=['jupyter', 'nbconvert', '--to', 'pdf', '--output',
'notebook_pdfs/submission_95.pdf', 'autogluon_each_location.ipynb'],
returncode=0)
```

[23]:
```
# feature importance
location="A"
split_time = pd.Timestamp("2022-10-28 22:00:00")
estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
estimated = estimated[estimated["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=estimated,␣
↪time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 30 features using
4392 rows with 10 shuffle sets… Time limit: 600s…
        2177.46s        = Expected runtime (217.75s per shuffle set)
        510.41s = Actual runtime (Completed 4 of 10 shuffle sets) (Early
stopping due to lack of time…)
```

[23]:

```
                               importance    stddev       p_value  n  \
direct_rad_1h:J              1.750273e+02  1.458075  7.967938e-08  4
clear_sky_energy_1h:J        9.819830e+01  1.661166  6.670618e-07  4
clear_sky_rad:W              9.457286e+01  1.677632  7.691624e-07  4
diffuse_rad_1h:J             7.852284e+01  1.083204  3.617592e-07  4
direct_rad:W                 7.113768e+01  1.634366  1.670680e-06  4
diffuse_rad:W                7.017501e+01  1.032625  4.390836e-07  4
sun_azimuth:d                5.092283e+01  1.988581  8.196862e-06  4
sun_elevation:d              3.993172e+01  1.285355  4.592624e-06  4
effective_cloud_cover:p      2.852983e+01  1.296365  1.290711e-05  4
total_cloud_cover:p          1.850211e+01  0.505266  2.805154e-06  4
is_in_shadow:idx             1.607287e+01  0.588508  6.757760e-06  4
t_1000hPa:K                  1.304928e+01  1.032875  6.796588e-05  4
snow_water:kgm2              1.298507e+01  0.628280  1.557989e-05  4
cloud_base_agl:m             1.289829e+01  0.359407  2.979969e-06  4
ceiling_height_agl:m         1.257206e+01  0.630674  1.736054e-05  4
relative_humidity_1000hPa:p  1.194747e+01  0.224948  9.196646e-07  4
wind_speed_10m:ms            1.105177e+01  0.661633  2.947868e-05  4
visibility:m                 1.093781e+01  0.731193  4.101209e-05  4
pressure_100m:hPa            8.875951e+00  0.844801  1.178800e-04  4
sfc_pressure:hPa             8.460762e+00  1.064410  2.705812e-04  4
msl_pressure:hPa             8.043555e+00  1.204036  4.531432e-04  4
fresh_snow_6h:cm             7.206400e+00  0.465633  3.704233e-05  4
pressure_50m:hPa             7.123986e+00  0.716822  1.391465e-04  4
is_day:idx                   6.574952e+00  0.470773  5.036246e-05  4
precip_type_5min:idx         5.773998e+00  0.909358  5.266406e-04  4
super_cooled_liquid_water:kgm2  4.341392e+00  0.687591  5.354706e-04  4
fresh_snow_3h:cm             3.754338e+00  0.710455  9.047694e-04  4
snow_depth:cm                2.751000e+00  0.355267  2.924570e-04  4
fresh_snow_1h:cm             2.620453e+00  0.391533  4.506794e-04  4
is_estimated                -3.115944e-08  0.000000  5.000000e-01  4

                                 p99_high       p99_low
direct_rad_1h:J              1.792855e+02  1.707691e+02
clear_sky_energy_1h:J        1.030497e+02  9.334694e+01
```

```
clear_sky_rad:W                    9.947231e+01  8.967342e+01
diffuse_rad_1h:J                   8.168629e+01  7.535939e+01
direct_rad:W                       7.591077e+01  6.636459e+01
diffuse_rad:W                      7.319075e+01  6.715927e+01
sun_azimuth:d                      5.673039e+01  4.511526e+01
sun_elevation:d                    4.368554e+01  3.617790e+01
effective_cloud_cover:p            3.231581e+01  2.474386e+01
total_cloud_cover:p                1.997772e+01  1.702650e+01
is_in_shadow:idx                   1.779158e+01  1.435416e+01
t_1000hPa:K                        1.606574e+01  1.003281e+01
snow_water:kgm2                    1.481993e+01  1.115020e+01
cloud_base_agl:m                   1.394792e+01  1.184866e+01
ceiling_height_agl:m               1.441391e+01  1.073020e+01
relative_humidity_1000hPa:p        1.260442e+01  1.129052e+01
wind_speed_10m:ms                  1.298404e+01  9.119498e+00
visibility:m                       1.307322e+01  8.802391e+00
pressure_100m:hPa                  1.134315e+01  6.408747e+00
sfc_pressure:hPa                   1.156932e+01  5.352200e+00
msl_pressure:hPa                   1.155989e+01  4.527223e+00
fresh_snow_6h:cm                   8.566261e+00  5.846539e+00
pressure_50m:hPa                   9.217432e+00  5.030540e+00
is_day:idx                         7.949822e+00  5.200081e+00
precip_type_5min:idx               8.429736e+00  3.118260e+00
super_cooled_liquid_water:kgm2     6.349471e+00  2.333314e+00
fresh_snow_3h:cm                   5.829189e+00  1.679486e+00
snow_depth:cm                      3.788540e+00  1.713459e+00
fresh_snow_1h:cm                   3.763906e+00  1.477000e+00
is_estimated                      -3.115944e-08 -3.115944e-08
```

[ ]:
```
# feature importance
observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
observed = observed[observed["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=observed,␣
↪time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 30 features using
5000 rows with 10 shuffle sets… Time limit: 600s…
        2393.51s        = Expected runtime (239.35s per shuffle set)
```

[ ]:
```
display(Javascript("IPython.notebook.save_checkpoint();"))
time.sleep(3)

subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
↪join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),␣
↪"autogluon_each_location.ipynb"])
```

```
[ ]: # import subprocess

     # def execute_git_command(directory, command):
     #     """Execute a Git command in the specified directory."""
     #     try:
     #         result = subprocess.check_output(['git', '-C', directory] + command,␣
      ↪stderr=subprocess.STDOUT)
     #         return result.decode('utf-8').strip(), True
     #     except subprocess.CalledProcessError as e:
     #         print(f"Git command failed with message: {e.output.decode('utf-8').
      ↪strip()}")
     #         return e.output.decode('utf-8').strip(), False

     # git_repo_path = "."

     # execute_git_command(git_repo_path, ['config', 'user.email',␣
      ↪'henrikskog01@gmail.com'])
     # execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
      ↪not None else 'Henrik eller Jørgen'])

     # branch_name = new_filename

     # # add datetime to branch name
     # branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

     # commit_msg = "run result"

     # execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

     # # Navigate to your repo and commit changes
     # execute_git_command(git_repo_path, ['add', '.'])
     # execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

     # # Push to remote
     # output, success = execute_git_command(git_repo_path, ['push',␣
      ↪'origin',branch_name])

     # # If the push fails, try setting an upstream branch and push again
     # if not success and 'upstream' in output:
     #     print("Attempting to set upstream and push again...")
     #     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
      ↪'origin',branch_name])
     #     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

     # execute_git_command(git_repo_path, ['checkout', 'main'])
```